LASER INTERFEROMETER GRAVITATIONAL WAVE OBSERVATORY

LIGO Laboratory / LIGO Scientific Collaboration

LIGO-T080188-v2 *ADVANCED LIGO* 6 January 2012

Models of the Advanced LIGO Suspensions in MATLAB™

Mark Barton, Calum Torrie, Ken Strain, Norna Robertson

Table of Contents

1 Introduction

1.1 Purpose and Scope

This document explains the physical assumptions, internal structure and usage of the models of the Advanced LIGO suspensions originally written in MATLAB by Calum Torrie, Ken Strain et al., and now maintained by Mark Barton.

1.2 References

P000039-00, Matthew Husman, PhD Thesis, 1999, University of Glasgow.

P000040-v1, Calum Torrie, PhD Thesis, 2000, University of Glasgow

T050255-07, Investigation into blade torsion, blade lateral flexibility, and the effect they have on blade and wire performance, I. Wilmut et al.

T020205-02, Suspension models in Mathematica, M.A. Barton

T020011-00, Suspension model comparisons, M.A. Barton.

T070101-00, Dissipation dilution, M.A. Barton.

T080096-00, Flexure corrections, M.A. Barton.

T010103-05, Advanced LIGO Suspension System Conceptual Design, N. Robertson, et al.

1.3 Version history

8/11/08: First draft, as T080188-00.

6/24/11: Second draft, as T080188-v1draft

7/28/11: -v1.

1/6/12: -v2. Upgraded symbexport3latfull.m in associated code archive from placeholder to substantive version.

2 History

2.1 Initial triple model by Calum Torrie

For his PhD thesis, Calum Torrie created a Matlab model of the GEO triple suspension. The suspension to be modeled consisted of (from the top down):

- Two blade springs
- Two wires (one per blade)
- A top mass with four blade springs
- Four wires (one per blade)
- An intermediate mass
- Four wires

- The optic

This is the same generic structure as the aLIGO BS/FM, HSTS and HLTS. It is pictured in Figure 1, taken from the suspension conceptual design, T080187, and Figure 2, taken from the output of the equivalent Mathematica model. To make this tractable for a hand derivation, a number of approximations were made:

- The 18 DOFs of the three masses were partitioned into four subsystems, which for symmetrical designs should be orthogonal: longitudinal/pitch (6 DOFs), transverse/roll (6), yaw (3), and vertical (3).
- The blades were allowed for by adding the compliance of the blades to that of the associated wires. This is an essentially perfect approximation if the working direction of the blades is aligned with that of the wires, and was still quite workable for the GEO design, which had the blade tips moving vertically but the wires at a modest diagonal.
- Four wires were assumed at each level in the derivation to maximize the similarity between the stages, and the fact that there are only two wires at the top level was allowed for by setting the front/back separation to zero there (`Su=0`).
- The wire bending stiffness was neglected in the Matlab model, but allowed for later by making corrections to the "d" values en route to manufacture (see T080096-00).
- The blade lateral compliance (see T050255-07, T080096-00) was not included, but fortunately was not significant for the parameters settled on.

- **Figure 1: Geometry of triple pendulum models – front and side views (from T080187)**

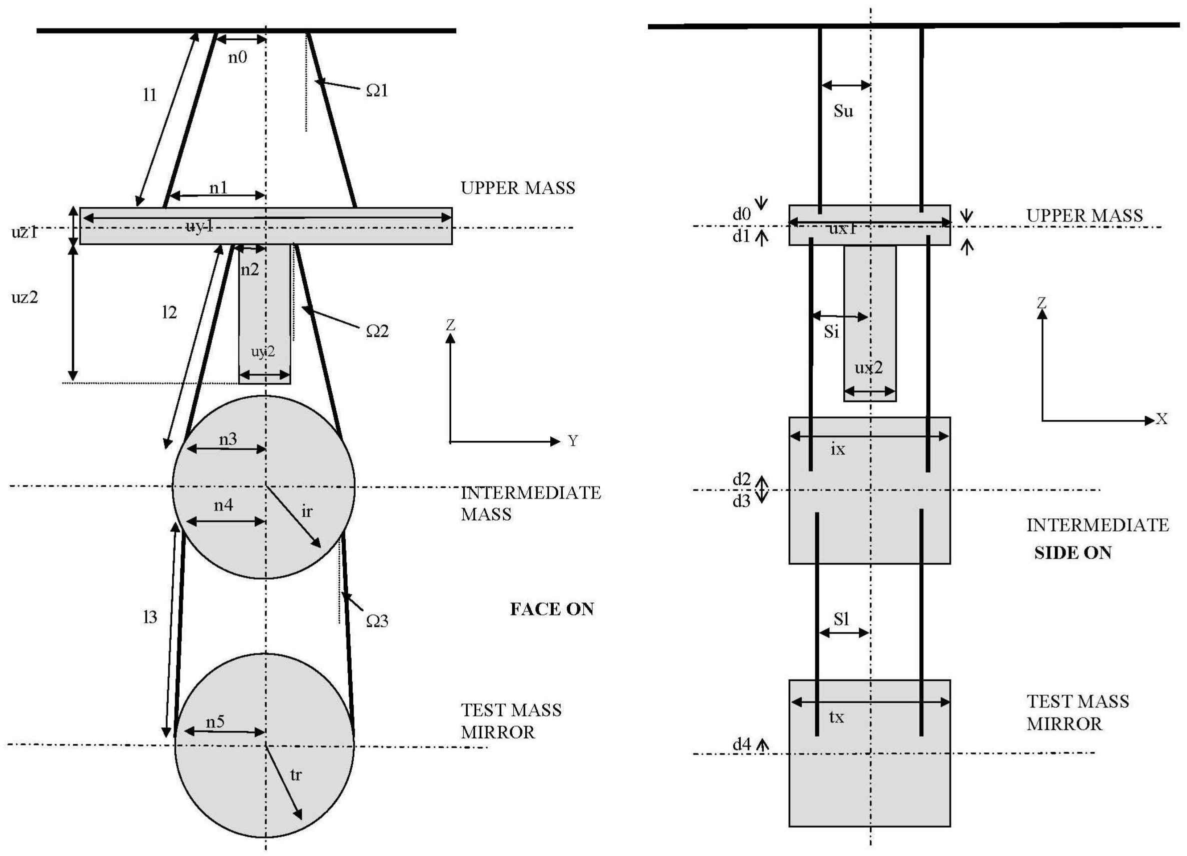
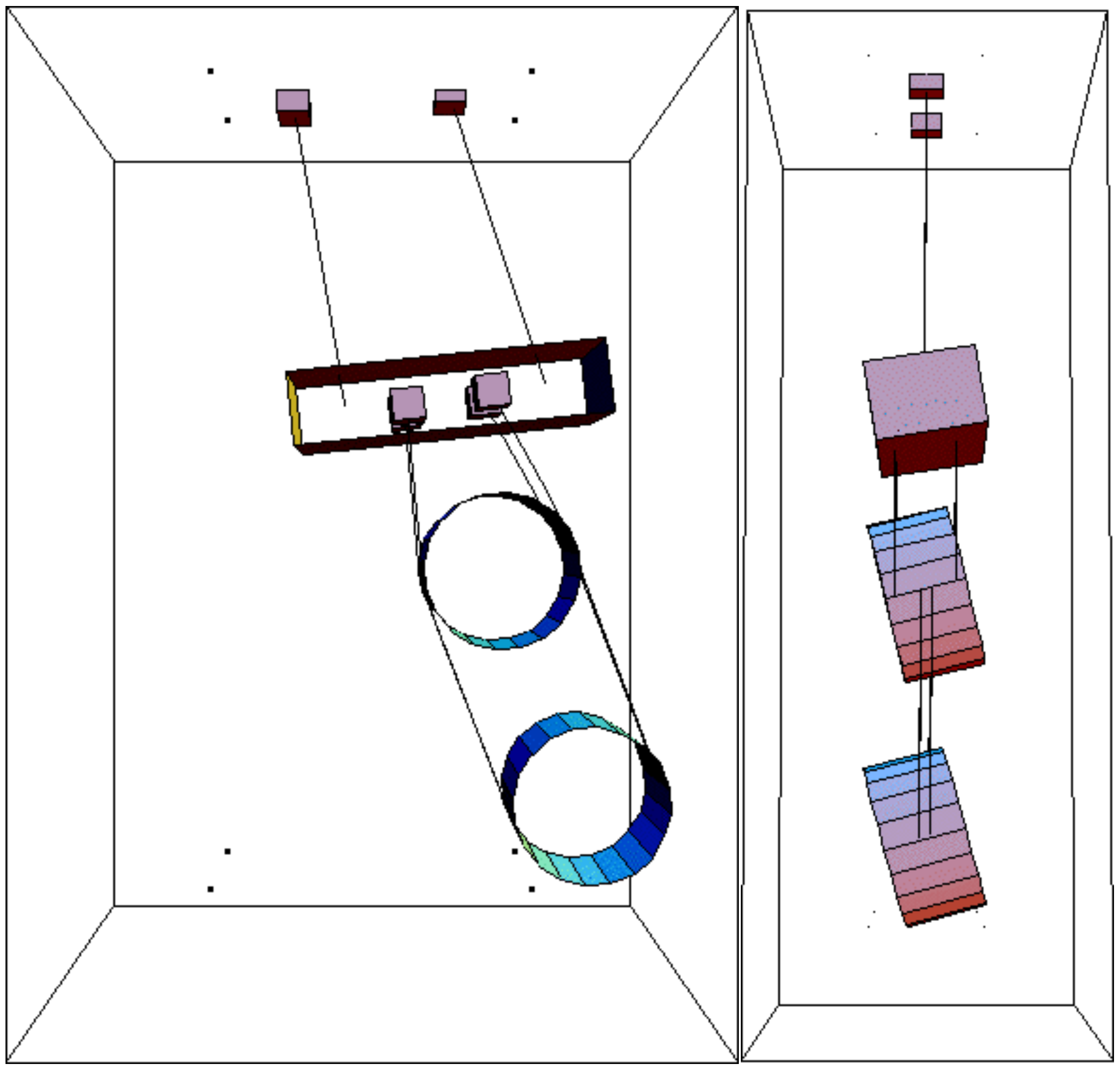


Figure 2: More realistic schematic of the triple (output of eigenplot[] in the Mathematica model). The blade spring tips are represented by small boxes.



2.2 Initial quad model by Ken Strain et al.

Ken Strain et al. later generalized the triple model to an aLIGO-style quad (Figure 3) by strategic copying, pasting and renaming of matrix elements and parameters to add a "new" top mass:

- Two blade springs
- Two wires (one per blade)
- A top mass with two blade springs (`mn`)
- Four wires (two per blade)
- An "upper-intermediate" mass with two blade springs (`m1` – cf. triple top mass)
- Four wires (two per blade)

- A “penultimate” mass (`m2`– cf. triple intermediate mass)
- Four wires
- The optic (`m3`– cf. triple optic)

The above generic structure is pictured in Figure 3, taken from the suspension conceptual design, T010103-05, and Figure 4, taken from the output of the Mathematica model. As with the initial version of the triple model, the blades were allowed for by corrections to the wire compliances and do not appear explicitly. A fuller listing of parameters is given in Table 1.

Because in the quad it is no longer the case that there is one wire per blade in all cases, the strategy for including the effect of the blades had to be adjusted. In the transverse/roll, yaw, and vertical subsytems the blade and wire compliances were added as previously. However in longitudinal/pitch there are several modes crucially involving differential stretching of the two wires on a single blade tip. Fortunately none of the longitudinal/pitch modes have significant excursion of the blades themselves, so a good approximation was to take the blades to have zero compliance for this subsystem only.

Figure 3: Geometry of quad pendulum models (from T010103).

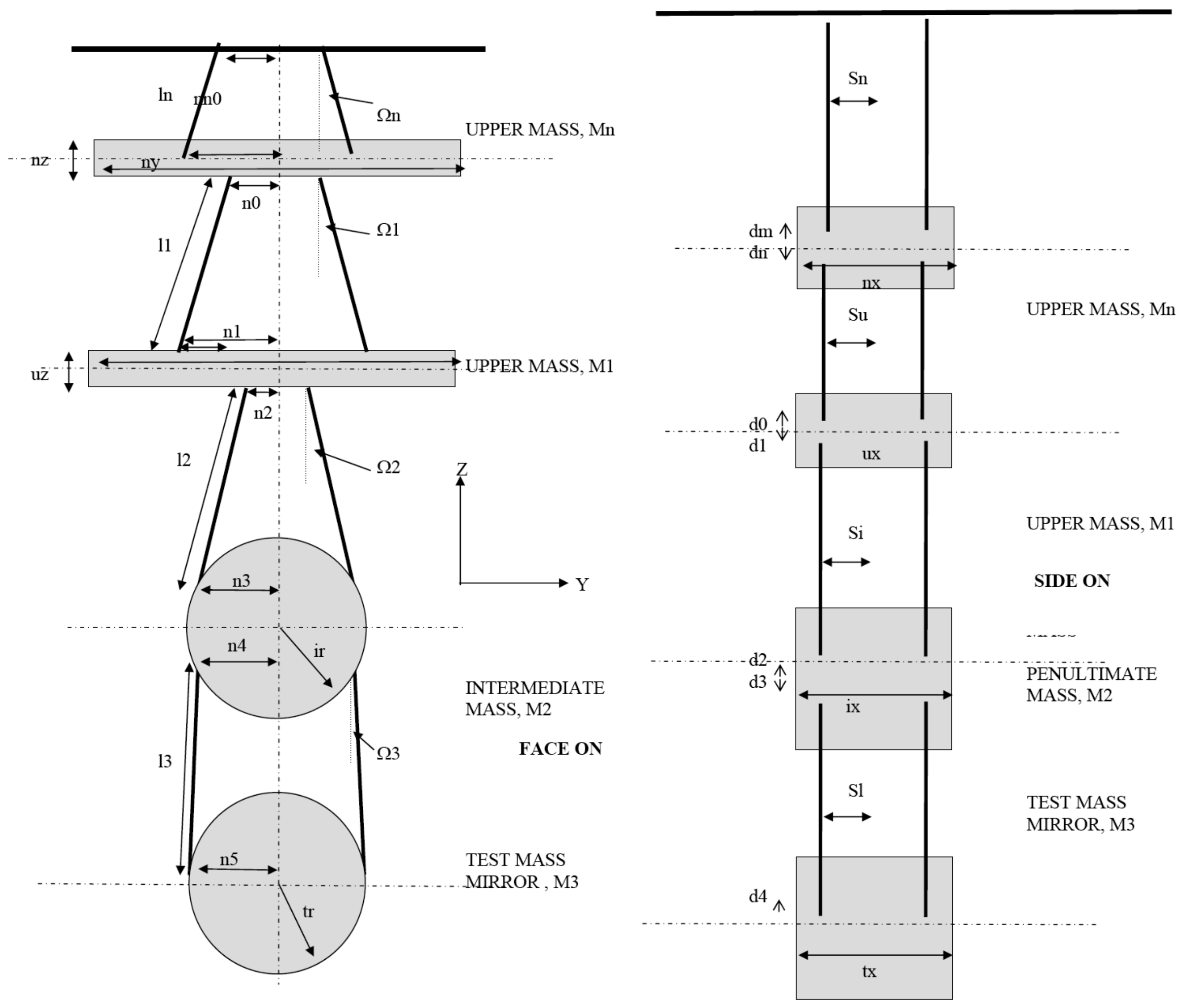
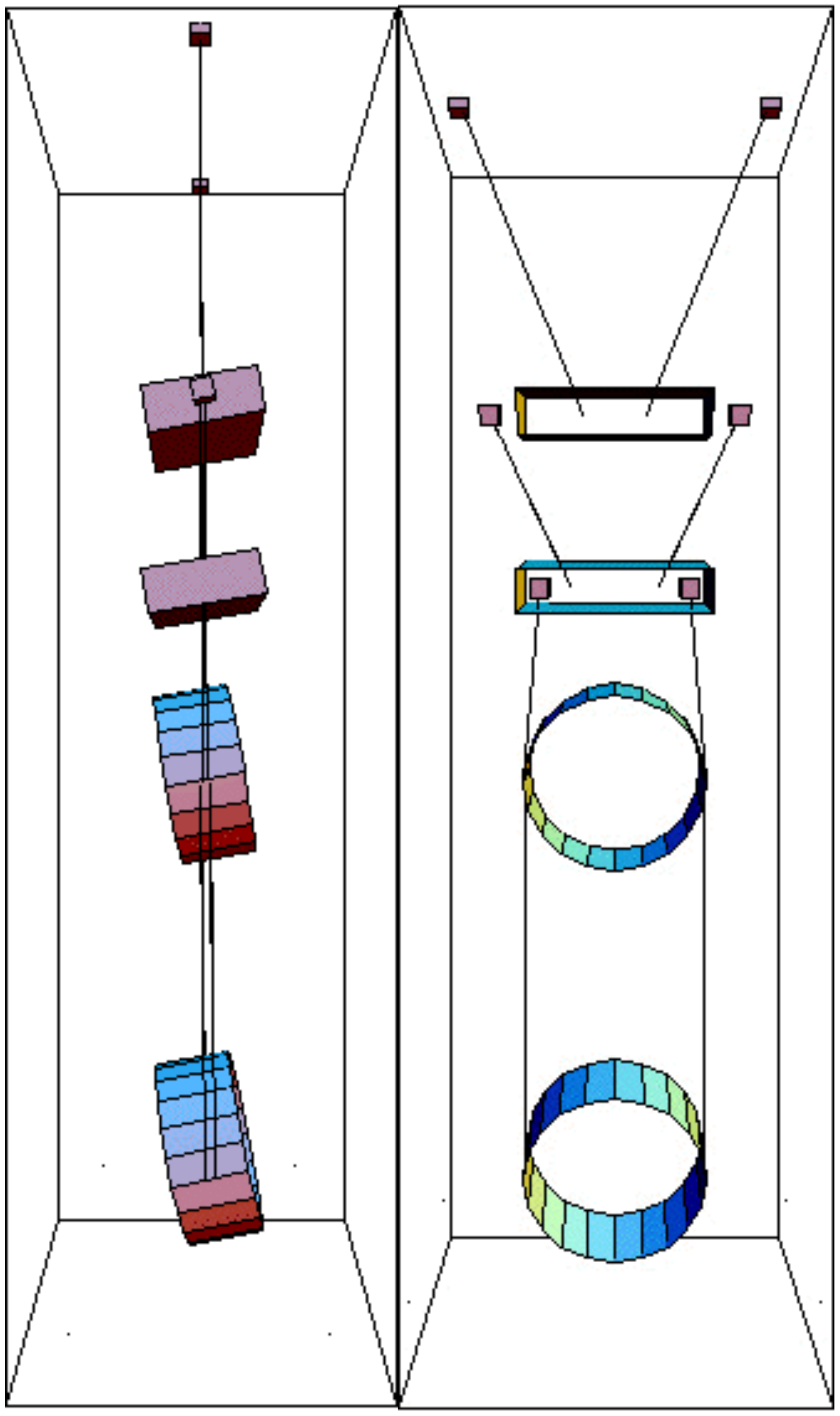


Figure 4: More realistic schematic of the quad (output of eigenplot[] from the Mathematica model). The blade spring tips are represented by small boxes.



2.3 Numerical validation against the Mathematica models

As a check on the Matlab, equivalent models were created in Mathematica using a pendulum modeling toolkit developed by Mark Barton (T020205). Parameter names were chosen to match as far as convenient, the main exception being in the stiffness of the wires and blades. As noted above, the Matlab models had adopted a convention of specifying these per side (whether there were two or four wires per side). Since wires and springs had to be modeled individually and explicitly in Mathematica, this was no longer natural. Rather than having different interpretations of identically named symbols, the names for wire and blade stiffness were deliberately chosen not to match. See Table 1 for the correspondences.

The Matlab models were initially validated numerically against the Mathematica models as described in T020211.

The Mathematica models explicitly included blades from the beginning, and the code included a switch to make the working direction of the blades either vertical (as in the physical system) or in line with the associated wires, as assumed for convenience in the Matlab. With the angled blades, the agreement with the Matlab was essentially perfect (three-figure or better agreement in the mode frequencies and shapes), validating both code bases.

2.4 Symbolic validation against the Mathematica models

Later (2005) the validation was redone exactly, by generating symbolic versions of the matrix elements in Mathematica, importing the ones from from Matlab and comparing them using Mathematica's symbolic algebra features. Calum's original code was shown to be exactly correct relative to the assumptions and approximations used to derive it, but some minor errors had crept into the working version of the triple and into the quad, and these were corrected.

2.5 Incorporation of Mathematica-derived code

Since the Mathematica could just as easily generate matrix elements for the improved blade model with the working direction vertical and export them as Matlab code, the opportunity was taken to swap these in. The implementation for the triple is as follows (the quad is very similar):

- The equivalent Mathematica model (TripleLite2) is intercepted at the point where numerical partial derivatives of the kinetic and potential energies are taken to produce the kinetic (mass) and potential (stiffness) matrices, and diverted to produce symbolic results instead.
- Substitutions are made for a few symbols that have different names or interpretations in the Matlab (see Table 2). (Because reckoning the elasticity per side was unnatural in the Mathematica where each wire and blade is specified individually, the elasticities were deliberately given different names rather than having quantities with the same name but different interpretations.)
- Symbolic versions of the kinetic matrix (km) and key sub-blocks of the master potential matrix (qm, xm, cqsm, cqxm, cxsm) are exported in Matlab syntax to a file with a name like symbexport3.m. (See Section 2.2, especially 2.2.2ff of T020205-v2 for the significance of these matrices.)
- In Matlab, the command file ssmake3MB.m loads `triplep.m` to define numerical values of parameters as fields of a structure `pend` and then runs `symbexport3.m` to get

numerical versions of km, qm, xm, etc. It assembles these into state space matrices, permuting rows and columns in some cases where the canonical order in the Mathematica (x/y/z/yaw/pitch/roll/…) differs from that in the Matlab. The reason for this division of labour is that the final steps involve taking some matrix inverses which could in principle be done symbolically on the Mathematica side but which would create inconveniently large expressions.

- The state space matrices are used to calculate the eigenmodes and eigenfrequencies, and to do time-domain simulation with Simulink, as with the old code.

Since this reworking, the Matlab code for both the quad and triple models has been dependent on the Mathematica, and has inherited all major improvements made on the Mathematica side. The only feature of the older Matlab models not supported in new ones is that the number of wires at the top level (`nwn` for quad and `nw1` for triple) is hardwired to two. The front-to-back wire separation at the top level (`sn` or `su`) is ignored.

Most of the models used to derive matrix elements as well as the actual derivations are published on the aLIGO SUS SVN repository at

https://redoubt.ligo-wa.caltech.edu/websvn/listing.php?repname=sus& .

See the associated wiki for instructions:

https://awiki.ligo-wa.caltech.edu/aLIGO/Suspensions/MathematicaModels .

Within each model (TripleLite2 or the like), the derivation of the matrix elements is done in a pseudo-case MatrixElementExport in the mark.barton folder, e.g.,

^/trunk/Common/MathematicaModels/TripleLite2/mark.barton/MatrixElementExport

where "^" is the repository root. For all of the published models there is provision for exporting the parameter set that defines a case of the Mathematica model as a Matlab command file usable with the associated Matlab model. The default case of each model comes with sample code, e.g.,

^/trunk/Common/MathematicaModels/TripleLite2/default/stdcalc/ASUS3L2ModelCalcExport.nb

generates

^/trunk/Common/MathematicaModels/TripleLite2/default/stdcalc/TripleLite2_default_triplep.m

2.6 Additional features developed in Mathematica and transplanted

2.6.1 Explicit support for wire bending stiffness

Early versions of the Matlab code ignored the bending stiffness of the wires. This was known from the beginning to be a significant effect, which particularly affects the lower pitch frequencies. It was allowed for by designing in terms of an ideal pendulum with flexible wires and then making corrections in the detailed mechanical design. Since the effective flexure point of the wire is a calculable distance inset from the attachment points, the required correction is to move the wire attachment points out by this amount at each end to make the effective flexure point correspond to the ideal model.

Because the effect of the wires amounts to a simple change of position of the wire attachment points, it turns out to be fairly simple to include this in the Matlab. The approach was developed in

Mathematica and tested against the "Stage 2" calculation which includes wire bending potential terms. The "d" and "n" distances (vertical and horizontal wire attachment positions) are adjusted to move the wire attachment points in along the direction of the wires by a flexure length at each end. The "l" values (wire lengths) are shortened to compensate for the inset attachment points and keep the overall dimensions the same, but the wire stretching elasticities are calculated from the unadjusted lengths. With these few subtleties ironed out, the agreement is essentially perfect - typically four significant figures or better in the mode frequencies. See T080096.

A switch `pend.stage2` enables this. If it is defined and non-zero, the geometry defined in the parameter file is interpreted as physical attachment points and transformed to effective values for ideally flexible wire as above before the matrix elements are evaluated.

Note however that there is a very slight known error in the B matrices generated with `pend.stage2`. Because there is no d value corresponding to the attachment point on the top blades, there is nothing to increase to reflect the fact that the effective flexure point is a few mm below the top blades. For the sake of discussion let us consider the quad and retrospectively call this quantity `dl` (since we are adding it before `dm` and `dn`). It makes exactly no difference to the A matrix (which describes the pendulum internal dynamics and determines the mode frequencies) because not changing it along with all the other dimensional changes to mimic wire stiffness is tantamount to simply hanging the pendulum at a different height. But `dl` does make a difference to the B matrix (which describes the effect of external inputs) because pitch and roll of the structure are reckoned about the line through the blade tips. Thus the main effect is to throw away a small cross-coupling from pitch of the structure due to the fact that the true effective flexure point is a few mm below the pitch axis. This would be easy to fix if it ever proved important, although it would require backwardly incompatible changes to multiple Matlab and Mathematica models. Note that this neglected effect is not the difference between zero and non-zero coupling from pitch of the structure – there is another coupling due to the way the top blades curl up when their working direction is pitched and no longer in the plane of the top wires, but it is smaller again.

2.6.2 Support for blade lateral compliance

Flexure of the blades in the "lateral" direction (i.e., at right angles to the length of the blade) turns out to be a significant effect in the quad. It is roughly equivalent to a reduced (more negative) "d" distance at the associated wire attachment point and lowers the fundamental pitch frequency. If not corrected, by increasing one or more of the d's, it can easily push the system into instability.

The effect was modeled exactly in Mathematica, and an alternative set of Matlab matrix elements was exported for each of quad and triple: `symbexport4lat.m` and `symbexport3lat.m`. They are used if the parameter file defines values for any of the lateral compliance fields of the `pend` data structure: `pend.kxn`, `pend.kx1` or `pend.kx2`. See Table 3 for the correspondence between Mathematica and Matlab names.

2.6.3 Dual blade triple model

At one point a triple design with only two blades at the top mass was considered (i.e., the blade count was 2+2 rather than 2+4). To support this, additional sets of matrix elements for dual blades with and without blade lateral compliance were derived: `symbexport3db.m` and `symbexport3dblat.m`. One or other of these is activated if the switch `pend.db` is set to a non-zero value in the triple parameter file, depending on whether blade lateral compliances are also defined (see above).

2.6.4 "Full" versions

In the quad, the blade springs in the top and upper intermediate masses are rather long and need to be slightly angled to fit. The large clamps holding the blade bases are off the transverse centreline, which creates off-diagonal terms in the MOI tensors for those masses. This in turn means that the longitudinal/pitch and transverse/roll subsystems are no longer independent, but significantly cross-coupled. To accommodate this, an alternative set of matrix elements `symbexport4latfull.m` was prepared with all 24 DOFs in one big state space. Because the (lack of) partitioning of the DOFs affects many of the supporting utilities in a way that the variants discussed above do not, it is used with a separate ssmake file (`ssmake4pv2eMB4f.m` or the like) and a separate Simulink model (`pendf.mdl` or the like). There are typically two versions of the Simulink model, one that uses an LTI block with an enormous number of scalar inputs and outputs to encapsulate the state space and a "vector" version (`pendfv.mdl` or the like) that uses a smaller number of vector inputs and outputs to simplify the plumbing by grouping signals of the same type.

Later, for completeness, a `symbexport3full.m` was prepared, not because any significant asymmetry is suspected in the triples, but so that controller code based on `symbexport4latfull`.m could be more easily adapted. The files `symbexport3dbfull.m` and `symbexport3dblatfull.m` are currently dummy versions that generate an error message if invoked, but substantive versions could easily be made if they were ever required. The file `symbexport3latfull.m` was a dummy in -v1 of this document but a substantive version was created for -v2. See Table 3 for the names of the off-diagonal MOI terms that can be specified.

2.6.5 Double pendulum models

There are two independent versions of a Matlab model of a double pendulum (as for the aLIGO Output Mode Cleaner suspension). One, by Chris Cueva, is a cut-down version of an older triple model and lacks modern conveniences like explicit allowance for wire bending. It has been used to produce results for the OMS SUS final design report, T080104-00. A second, by Mark Barton is based on the modern code with a set of matrix elements from Mathematica exists but has not seen production use.

2.6.6 Other models

There are many other models that exist in Mathematica form and could be converted to Matlab with a day or two's work. These include a iLIGO-style single pendulum without blades, and an aLIGO TipTilt-style single pendulum with blades. Readers who would find Matlab versions of this or other models helpful should ask Mark Barton to do a conversion.

3 Code Archive

Associated with this document in the DCC is a code archive with historically important versions of the triple and quad models.

3.1.1 Calum Torrie Thesis Triple

This is the original model from Calum Torrie's thesis (P000040-v1). To use, run one of the parameter files, `ajt.m` or `jbr.m`.

3.1.2 Second Generation Triple

This is the original triple model as adapted by Ken Strain et al., based on Calum's matrix elements., with conveniences such as a Simulink wrapper model to do time domain simulation. To use, run `pend_ref.m` and/or `generate_simulink.m`.

3.1.3 Second Generation Quad

This is a quad model parallel to the second generation triple model above, created by Ken Strain et al., with matrix elements derived by cloning a level in Calum's triple matrix elements, and with the same conveniences as the second generation triple. To use, run `quad_ref.m` and/or `generate_simulink.m`.

3.1.4 Third Generation Triple

This is the triple model as reworked by Mark Barton, using matrix elements exported from Mathematica. It supports the same conveniences as the second-generation triple but includes a better blade model with vertically acting blades and options for explicit wire bending stiffness, blade lateral compliance, and double (rather than four) blades at the middle level. To work with it as four separate subsystems (longitudinal/pitch etc), run `pend_ref.m` and/or `generate_simulink.m`. To work with it as one large system, run `pend_ref_f.m` and/or `generate_simulink_f.m`.

3.1.5 Third Generation Quad

This is the quad model as reworked by Mark Barton, using matrix elements exported from Mathematica. It supports the same conveniences as the second-generation quad but includes a better blade model with vertically acting blades and options for explicit wire bending stiffness and blade lateral compliance. To work with it as four separate subsystems (longitudinal/pitch etc), run `quad_ref.m` and/or `generate_simulink.m`. To work with it as one large system, run `quad_ref_f.m` and/or `generate_simulink_f.m`.

3.1.6 Second Generation Double

This has been adapted by Chris Cueva from the second generation triple. To use, run `pend_ref_doublep.m` and/or `generate_simulink_doublep.m`.

3.1.7 Third Generation Double

This has been adapted by Mark Barton from the second generation triple. To work with it as four separate subsystems (longitudinal/pitch etc), run `double_ref.m` and/or `generate_simulink.m`. To work with it as one large system, run `double_ref_f.m` and/or `generate_simulink_f.m`

4 Tables

Table 1: Quad/triple pendulum nomenclature. The order within each group is always from the top down, with quantities with "3" in the name typically referring to the optic. Quantities in parentheses are for the quad only.

Symbol	Description
(nwn), nw1, nw2, nw3	numbers of wires (NOTE: for historical reasons these are treated as inputs, but setting them to anything but 2,4,4,4 (quad) or 2,4,4 (triple) in the Mathematica-derived code will give undefined results.)
(mn), m1, m2, m3	masses
(Inx), I1x, I2x, I3x	MOIs about x direction (i.e., for roll)
(Iny), I1y, I2y, I3y	MOIs about y direction (i.e., for pitch)
(Inz), I1z, I2z, I3z	MOIs about z direction (i.e., for yaw)
(kwn), kw1, kw2, kw3	wire stretching elasticities, per side (rather than per wire)
(dm), d0, d2, d4	vertical distances from the COMs of the respective masses up to the wire attachment points for wires coming down from above
(dn), d1, d3	vertical distances from the COMs of the respective masses up to the wire attachment points for wires coming down from above. (There is no d5, because nothing hangs down from the optic.)
(nn0, nn1), n0, n1, n2, n3, n4, n5	longitudinal distances from the centre to the wire attachment points
(sn), su, si, sl	transverse distances from the centre to the wire attachment points (NOTE: sn in the quad and su in the triple are ignored.)
(ln), l1, l2, l3	wire lengths (under tension)
(Yn), Y1, Y2, Y3	Young's moduli for the wires
kcn, kc1, kc2, kc3	blade stiffnesses in the working direction (per side rather than per blade)
Additional quantities used in some references or within the code	
(mn3), m13, m23	sum of all masses from the indicated stage down to the bottom
(Ωn), Ω1, Ω2, Ω3	wire angles from the vertical as viewed face on (+ve = wider at the bottom)
(cn), c1, c2, c3	cosines of Ωn,…Ω3

sin, si1, si2, si3	sines of Ωn,…Ω3 (+ve = wider at the bottom)
(flexn), flex1, flex2, flex3	wire flexure lengths
(tln), tl1, tl2, tl3	"true lengths", i.e, vertical distances from COM to COM
(An), A1, A2, A3	wire cross-sectional areas
(Mn1), M11, M21, M31	wire moments of area for bending in the longitudinal direction
(Mn2), M12, M22, M32	wire moments of area for bending in the transverse direction
(ufcn), ufc1, ufc2	"uncoupled frequencies", i.e., the resonant frequencies that would be obtained if each spring supported only its share of the stage immediately below

Table 2: Correspondences between blade and wire quantities in Matlab vs. Mathematica. Items in () only apply in models that include blade lateral compliance (usual for quad, but not triple).

Matlab	Mathematica
Quad	
kn	kwn
k1	2*kw1 (two wires per side)
k2	2*kw2 (two wires per side)
k3	2*kw3 (two wires per side)
kcn	kbuz
kc1	kbiz
kc2	kblz
(kxn)	(kbux)
(kx1)	(kbix)
(kx2)	(kblx)
Triple	
k1	kw1
k2	2*kw2 (two wires per side)
k3	2*kw3 (two wires per side)
kc1	kbuz

kc2	2*kblz (two blades per side)
(kx1)	(kbux)
(kx2)	(2*kblx) (two blades per side)

Table 3: Additional quantities used in extended versions of the models (see below)

Symbol	Description
kxn, kx1, kx2, kx3	blade stiffnesses in the lateral direction (per side rather than per blade)
(Inxy, Inyz, Inzx), I1xy, I1yz, I1zx, I2xy, I2yz, I2zx, I3xy, I3yz, I3zx	off-diagonal components of the MOI tensors for the masses